J. Lohn, W. Kraus, G. Haith, "Comparing a Coevolutionary Genetic Algorithm for Multiobjective Optimization," Proceedings of the 2002 IEEE Congress on Evolutionary Computation, May 2002, pp. 1157-1162.

Comparing a Coevolutionary Genetic Algorithm for Multiobjective Optimization

Jason D. Lohn[1], William F. Kraus[2], Gary L. Haith[3]
[1]Computational Sciences Division, MS 269-1
[2]QSS Group, Inc., MS 269-3
[1,2]NASA Ames Research Center, Moffett Field, CA, 94035 USA
[3]NAREX, Inc, Golden, CO, USA
{jlohn,bkraus}@email.arc.nasa.gov haith@stanfordalumni.org

Abstract— **We present results from a study comparing a recently developed coevolutionary genetic algorithm (CGA) against a set of evolutionary algorithms using a suite of multiobjective optimization benchmarks. The CGA embodies competitive coevolution and employs a simple, straightforward target population representation and fitness calculation based on developmental theory of learning. Because of these properties, setting up the additional population is trivial making implementation no more difficult than using a standard GA. Empirical results using a suite of two-objective test functions indicate that this CGA performs well at finding solutions on convex, nonconvex, discrete, and deceptive Pareto-optimal fronts, while giving respectable results on a nonuniform optimization. On a multimodal Pareto front, the CGA yields poor coverage across the Pareto front, yet finds a solution that dominates all the solutions produced by the eight other algorithms.**

I. Introduction

Cooperation and competition between populations of organisms in nature has inspired researchers to incorporate coevolutionary dynamics into genetic algorithms. The common element in these approaches is the inclusion of one or more additional populations. A growing body of research explores coevolutionary approaches that capitalize on this dynamic quality (for review, see [11]). This coevolutionary work has largely concentrated on competitive interactions. The interactions can be between individuals that compete in a symmetric game-like context [12], [14], or between populations of different types of individuals that compete in predator/prey type relationships [4], [9], [8], [5], [13]. In these cases, individuals are rewarded if they defeat the individuals with which they compete. These interactions can support "arms-races" in which the individuals force each other to become increasingly competent.

A few studies have investigated the role of cooperation and how it can help solve some problems endemic to evolutionary methods, like the difficulty of choosing an appropriate encoding for the individuals [10] and the difficulty of decomposing composite problems [1]. Other studies have found that a balance of cooperation and competition is necessary to prevent evolutionary algorithms from getting trapped in local minima, or "Mediocre Stable States" [2].

In this paper we describe a coevolutionary genetic algorithm (CGA) whose fitness calculations are inspired by developmental theory [3]. The fundamental idea is to use coevolutionary dynamics to automatically regulate the level of difficulty, from easy to hard, posed by a population of tests. We then describe multiobjective optimization problems and a suite of test functions that we use to judge the performance of the CGA. Empirical results from the CGA runs are presented and compared to previously-published results.

II. Coevolutionary GA

The coevolutionary algorithm we present is based on an algorithm used in previous evolvable hardware applications [6], [7], and is based on competition between two populations. The population of candidate solutions, or trial population, is represented and manipulated much the same as the main population in a standard genetic algorithm. The second population, or target population, consists of *target objective vectors* (TOVs) – vectors containing targets for the individual objectives to be optimized. An overview of the algorithm is presented in Figure 1.

The population of TOVs is used to encapsulate the level of difficulty that the trial population faces. Under the control of the genetic algorithm, the TOVs evolve from easy to difficult based on the level of proficiency of the trial population. The algorithm designer need only specify two TOVs: an easy TOV and a difficult TOV, the latter being the ultimate goal of the run (analogous to stopping criteria in standard evolutionary algorithms). The first generation of TOVs are randomly initialized with TOV values that fall within the "easy" and "difficult" bounds. The CGA seeds the easy TOVs into early generations of the run to guarantee that the coevolutionary dynamic will be used – as we shall see, if all TOVs were too difficult for generation zero individuals, there would be no competitive mechanism and hence no fitness feedback between populations.

Each TOV consists of a set of target objectives that act like thresholds: all thresholds must be met or exceeded in order for the TOV to be "solved," and hence gain fitness[1].

[1]This all-or-nothing property can be relaxed to accommodate partial solutions, however that version of the algorithm will be reported on in future work.

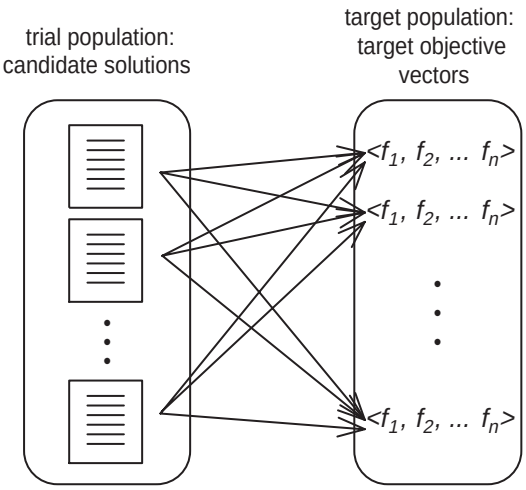


Fig. 1. Overview of coevolutionary genetic algorithm.

For example, assume a four-objective TOV where one desires to maximize the first objective and minimize the remaining objectives. Then, the TOV $\langle 63.0, 0.6, 0.8, 9.5\rangle$ solves $\langle 60.0, 1.0, 1.0, 10.0\rangle$, but $\langle 58.0, 0.6, 1.2, 18.0\rangle$ does not.

The general form of the fitness calculations are as follows. Trial individuals are rewarded for solving difficult TOVs. The most difficult TOV at a given generation is defined to be the one that only one trial individual can solve. Such a TOV garners the highest fitness score. TOVs that are unsolvable, or are very easy to solve by the current trial population, are given low fitness scores. Fitness of individual in the trial population is computed as follows. Individual i "plays" each TOV in the second population and a score, s_i, is computed:

$$s_i = \sum_{j \in \widehat{\text{tov}_i}} \frac{1}{\begin{array}{c}\#\text{ trial}\\ \text{individuals}\\ \text{that solve tov}_j\end{array}}$$

where $\widehat{\text{tov}_i}$ is the set of TOV indexes such that individual i solves tov_j. Note that the denominator in the above fraction is guaranteed to be greater than or equal to one due to the restriction on j. Then s_i is normalized linearly between its upper and lower bounds such that 0.0 is the best score and 1.0 the worst:

$$F(\text{trial individual}_i) = 1.0 - s_i/M_2$$

where M_2 is the size of the TOV population. The effect of s is to reward trial individuals that solve the more difficult TOVs. A TOV has the greatest difficulty level when exactly one trial individual can solve it. If many trial individuals can solve a particular TOV, the fitness contribution in s is shared among the trial individuals [13].

Fitness of an individual TOV is computed as follows. Let x_j denote the number of trial individuals that solve tov_j, and M_1 be the trial population size. The fitness is essentially x_j, scaled and normalized, with a tractability constraint:

$$F(\text{tov}_j) = \begin{cases} 1.0 & x_j = 0 \\ \frac{1}{(M_1 - 1)}(x_j - 1.0) & x_j \geq 1 \end{cases}$$

The tractability constraint gives a target vector a score of 1.0 (the "worst" score) when no trial individuals can solve it. This puts pressure on the TOV population to pose difficult, yet solvable problems to the trial population.

The target population is manipulated like the single population of a typical genetic algorithm. No explicit niching operators were used to enhance diversity. Constrained mutation was used to ensure TOVs remained valid after mutation. One point crossover was implemented by choosing cut-points between individual objective values.

In typical real-world applications, the running time of the evaluation function swamps out the running time of the underlying evolutionary algorithm. However, for comparison to other multiobjective EAs, we can compute the time complexity of the CGA as follows. Let the two populations be of size M_1 and M_2, and let n denote the number of objectives. Evaluation of the trial population runs at $O(nM_1)$. Then, fully-crossed pairwise comparisons (to determine which trial individuals solve which target populations) are required, running at $O(nM_1M_2)$. Thus the time complexity is $O(nM_1M_2)$ for the core of the CGA. In comparison, SPEA [16] and NSGA-II run at $O(nM^2)$, which is identical when $M_1 = M_2$ is used in the CGA.

III. Multiobjective Optimization

The notion of weighing tradeoffs is common to problems in everyday life, science, and engineering. Buying a less expensive product might tradeoff product quality for the ability to buy more of something else. Adding an additional science instrument to a spacecraft trades off increased costs for increased science return. Hard optimization problems typically require many decisions on the input side and many objectives to optimize on the output side. The set of objectives forms a space where points in the space represent individual solutions. The goal of course is to find the best or optimal solutions to the optimization problem at hand. *Pareto optimality* defines how to determine the set of optimal solutions. A solution is Pareto-optimal if no other solution can improve one objective function without a simultaneous deterioration of at least one of the other objectives. A set of such solutions is called the Pareto-optimal front. An example of a Pareto front is seen in Figure 2.

Evolutionary algorithms (EAs) have recently attracted much attention in the exploration of Pareto-optimal fronts. It is claimed that EAs are the preeminent search algorithms for such tasks [17]. An overview of EAs in multiobjective EAs can be found in [15].

Below we briefly touch on relevant terminology and definitions regarding multiobjective optimization problems

0-7803-7282-4/02/$10.00 ©2002 IEEE

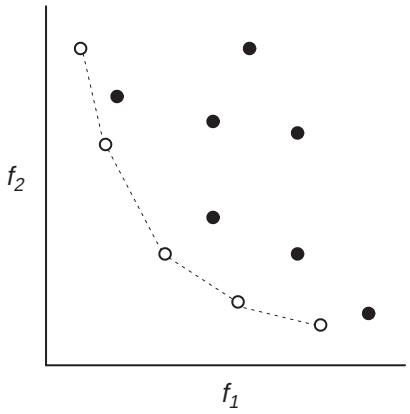


Fig. 2. Example of Pareto front when minimizing two objectives f_1 and f_2. Nondominated solutions are represented as hollow circles (○) and dominated solutions by filled circles (●).

(following [17]). The set of input parameters, or decision variable, is called the *decision vector*. The set of objective functions that measure the performance of the system is called the *objective vector*. In an evolutionary algorithm framework, a decision vector naturally corresponds to a candidate solution, and the functions comprising the objective vector are typically incorporated, by various techniques, into the fitness function(s).

A dominance test is a way to measure the relative performance among decision vectors. Given two decision vectors **a** and **b**, **a** *dominates* **b** if and only if **a** ties or exceeds **b**'s performance on every objective, and there exists at least one objective where **a**'s performance strictly exceeds **b**'s. Using this test, we can pare down any given set of decision vectors and find the set of nondominated decision vectors. Such a set is said to form the nondominated front. If the nondominated set resulted from testing every possible decision vector, then the nondominated set is the Pareto-optimal front.

A coverage test adds a test for equality to the dominance test. Given two decision vectors **a** and **b**, **a** *covers* **b** if and only if **a** dominates **b** or **a**'s objective vector is identical to **b**'s. The coverage test is used to compare two algorithms as follows. The function $\mathcal{C}(A, B)$ computes the percentage of algorithm B's solutions that are covered by solutions produced by A.

The above tests (see [17] for formal definitions) are used to assess the ability of algorithms to optimize a set of decision vectors. The dominance test will be used to cull dominated solutions produced by a given algorithm. The coverage test will be used to compare the solutions produced by algorithms head-to-head.

IV. Experimental Setup

We follow the suite of multiobjective test functions and empirical results presented in [17]. Briefly, there were seven multiobjective evolutionary algorithms and one random search algorithm executed on six test functions. The algorithms compared in [17] were: random search (RAND), Fonseca and Fleming's multiobjective GA (FFGA), the Niched Pareto GA (NPGA), Hajela and Lin's weighted sum approach (HLGA), the Vector Evaluated GA (VEGA), the Nondominated Sorting GA (NSGA), a single-objective EA using weighted-sum aggregation (SOEA), and the Strength Pareto GA (SPEA). The CGA described above is denoted COEV.

Number of generations	250
Trial population size	100
Target objective vector population size	100
Crossover rate (both populations)	0.8
Mutation rate (both populations)	0.01

TABLE I
Coevolutionary GA parameters.

The test functions, $\mathcal{T}_1$ – $\mathcal{T}_6$, were chosen because they provide a range of difficulties for multiobjective optimization (e.g., multimodality, deception, isolated optima). In each optimization, it is desired to minimize the objective vector $\langle f_1, f_2 \rangle$ by finding its Pareto-optimal front.

To allow a direct comparison to the results in [17], we followed the run setup as closely as possible: thirty CGA runs were executed for each test function using the parameters shown in Table I. To compute the nondominated front for the CGA, we did the following. For each CGA run, we collected all the output objective vectors ($\langle f_1, f_2 \rangle$) corresponding to the individuals evaluated during the run. For each test function, the output objective vectors from five randomly-selected runs were combined and a domination test removed all the dominated solutions. For the algorithm-to-algorithm coverage test (function $\mathcal{C}$ described above), we used the results from the thirty runs as follows. The nondominated set from each run was computed. Then the domination test was performed by pitting the nondominated set from algorithm A, run i, against the nondominated set from algorithm B, run i. Statistics, in the form of boxplots (described below), were computed using the resulting thirty $\mathcal{C}$ values. Both $\mathcal{C}(A, B)$ and $\mathcal{C}(B, A)$ were computed as they may be different.

V. Results

As noted in the literature, comparing multiobjective optimization algorithms against each other can be difficult. One would like an algorithm to minimize the distance to the Pareto-optimal front and provide uniform coverage of the Pareto-optimal front for a wide range of values. Thus, comparisons become multiobjective optimization problems themselves: is an algorithm that finds a handful of Pareto-optimal solutions better than an algorithm that finds a wide, uniform distribution of near Pareto-optimal solutions? With this in mind we present

the experimental results.

Figures 3–4 show the results from the six test functions. On each graph the optimal Pareto front is drawn as a curve, data points for the eight comparison algorithms are shown in gray[2], and the data points from the CGA runs (COEV) are shown as black circles.

In general, the results show that the CGA is a relatively strong performer: it always exceeds random search and has qualitatively good performance against strong algorithms such as SPEA and NSGA. On the first two test functions, the CGA has the qualitatively best distribution and alignment to the Pareto-optimal curve. In the third test function, it performs on par with SPEA. In the fourth test function, a multimodal surface, the CGA has poor coverage, yet find a solution that dominates nearly all the others. On the deceptive test function, $\mathcal{T}_5$, the CGA provides relatively excellent coverage except at low f_1 values, with SOEA doing better there. On the nonuniform test function, $\mathcal{T}_6$, SPEA is the only algorithm to find any Pareto-optimal solutions, and is able to span the width of the front. However CGA provides near-optimal solutions, with good coverage at high values of f_1.

The head-to-head algorithm comparisons using the $\mathcal{C}$ metric are shown in the boxplots of Figure 5. Each boxplot contains results from each of the six test functions: the dark dash is the median, the the top of the box is the upper quartile, the bottom of the box is the lower quartile. As can be seen, on all test functions the solutions found by the CGA statistically cover the solutions found by RAND, FFGA, NPGA, HLGA, and VEGA. The CGA's weakest results are on $\mathcal{T}_6$ against NSGA, SOEA, SPEA.

VI. Conclusion

Multiobjective optimization is clearly one of the most important class of problems in science and engineering. Solution techniques that are effective at searching what are typically vast search spaces, and finding a selection of Pareto-optimal solutions are very desirable. In this paper we presented a coevolutionary genetic algorithm inspired by development learning theory, and compared it empirically to seven other evolutionary search techniques for multiobjective optimization. In terms of algorithm design, CGA is no more difficult to design and implement than a typical genetic algorithm. In fact, because the fitness functions are identical across application domains, implementation may be viewed as being easier. The results show that the CGA performed very well compared to the other evolutionary algorithms and random search. On four of the six functions, it could be argued that the CGA qualitatively performed on par with or outperformed the other algorithms. Missing from this study is a comparison against traditional optimization algorithms, which we leave for future work.

[2]Data used to plot the curves from the non-coevolutionary runs was obtained from the authors of [17]. Differences between the curves in this paper and in [17] can be seen because the data was sampled randomly.

VII. Acknowledgments

This research was sponsored by the NASA Intelligent Systems Program.

References

[1] K. DeJong, M. Potter, Evolving Complex Structures via Cooperative Coevolution. *Proc. Fourth Annual Conf. on Evolutionary Programming* MIT Press, pp. 307-317, 1995.

[2] S.G. Ficici, Challenges in coevolutionary learning: Arms-race dynamics, open-endedness, and mediocre stable states, 1995.

[3] G.L. Haith, S.P. Colombano, J.D. Lohn, D. Stassinopoulos, "Coevolution for Problem Simplification," *Proc. 1999 Genetic and Evolutionary Computation Conference,* 1999.

[4] D. W. Hillis, Co-evolving parasites improve simulated evolution as an optimization procedure. Pages 313-324 of: Langton, C., Taylor, C., Farmer, J. D., Rasmussen, S. (eds), Artificial life 2, vol. X. Redwood City, CA: AddisonWesley, 1991.

[5] H. Juille, J. Pollack, "Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules," *Proc.Third Annual Genetic Programming Conf.*, 1998.

[6] J.D. Lohn, G.L. Haith, S.P. Colombano, D. Stassinopoulos, "A Comparison of Dynamic Fitness Schedules for Evolutionary Design of Amplifiers," Proc. of the First NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA, IEEE Computer Society Press, 1999, pp. 87-92.

[7] J.D. Lohn, W.F. Kraus, D.S. Linden, S.P. Colombano, "Evolutionary Optimization of Yagi-Uda Antennas," Proc. of the Fourth International Conference on Evolvable Systems, Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, M. Yasunaga (Eds.), Tokyo, October 3-5, 2001, Springer-Verlag, pp. 236-243.

[8] J. Paredis, Coevolutionary constraint satisfaction. Pages 46-55 of: Proceedings of the third international conference on parallel problem solving from nature, vol. 866. Springer-Verlag, 1994.

[9] J. Paredis, Steps towards co-evolutionary classification neural networks. Pages 102-108 of: Brooks, R., Maes, P. (eds), Artificial Life IV. Cambridge, MA: MIT Press, 1994.

[10] J. Paredis, The symbiotic evolution of solutions and their representations. Pages 359-365 of: Eshelman, L. (ed), Proceedings of the sixth international conference on genetic algorithms. San Mateo, CA: Morgan Kaufmann, 1995.

[11] J. Paredis, The handbook of evolutionary computation. Oxford University Press. Chap. Coevolutionary Algorithms, 1998.

[12] J. Pollack, A. Blair, M. Land, "Coevolution of a backgammon player," *Proc. of Artificial Life 5,* Langton, C. (ed), MIT Press, 1996.

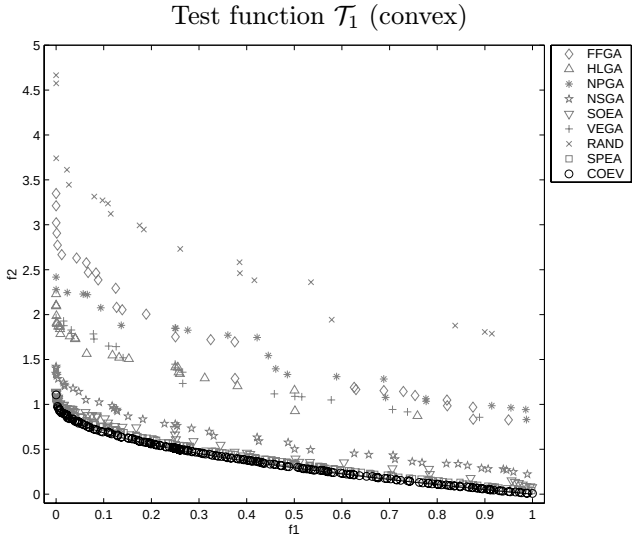
[13] C.D. Rosin, R.K. Belew, *New Methods for Competitive Coevolution,* Tech. Rept. CS96-491, Department of Computer Science and Engineering, University of California, San Diego, 1996.

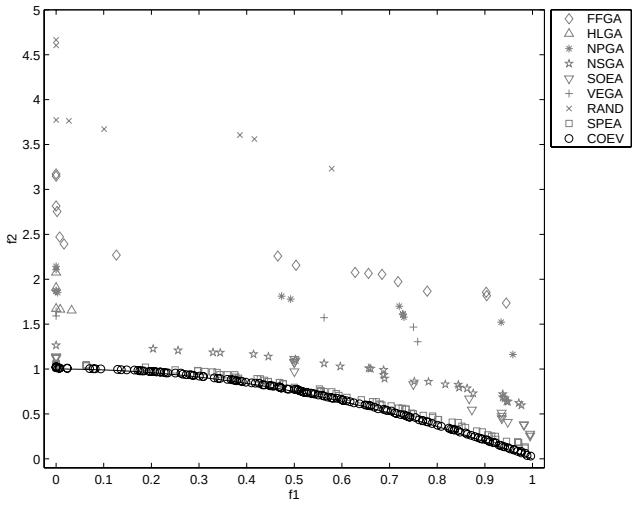
[14] C.D. Rosin, "Coevolutionary Search Among Adversaries," Ph.D. Thesis, University of California, San Diego, 1997.

[15] D. Van Veldhuizen, "Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations," Ph.D. Thesis, Air Force Institute of Technology, 1999.

[16] E. Zitzler, L. Thiele, "Multiobjective Optimization using Evolutionary Algorithms – A Comparitive Case Study," *Parallel Problem Solving From Nature V,* pp. 292-301, Springer, 1998.

[17] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evolutionary Computation*, 8(2), pp. 173-195, Summer 2000.





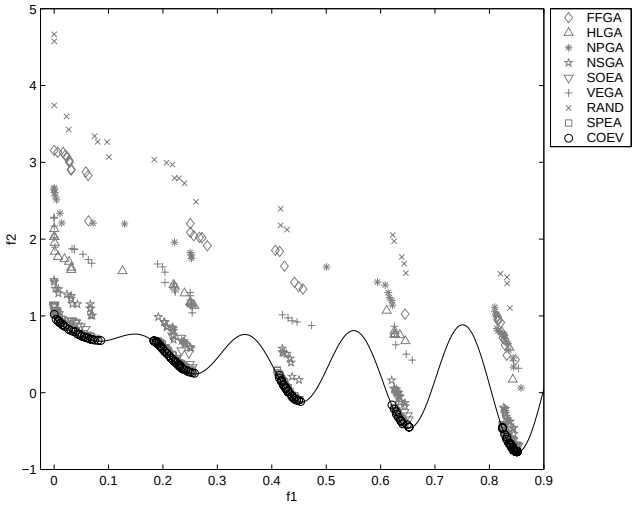
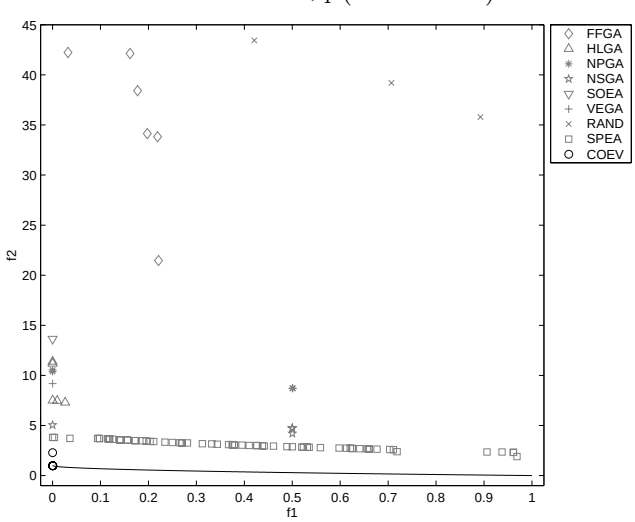
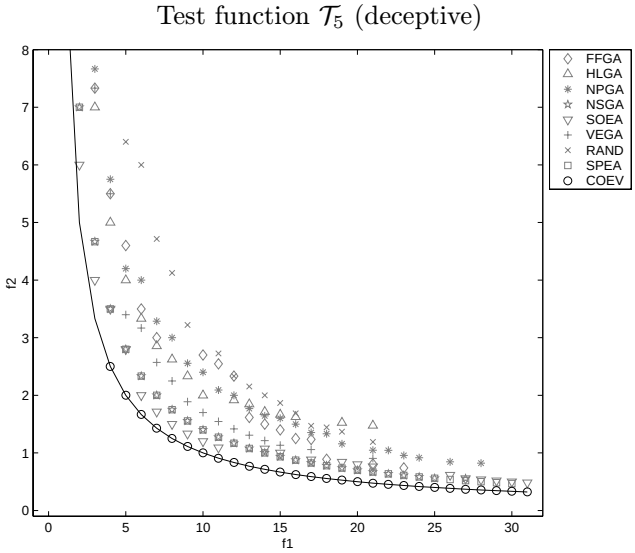


Fig. 3. Test functions $\mathcal{T}_1$ – $\mathcal{T}_3$.





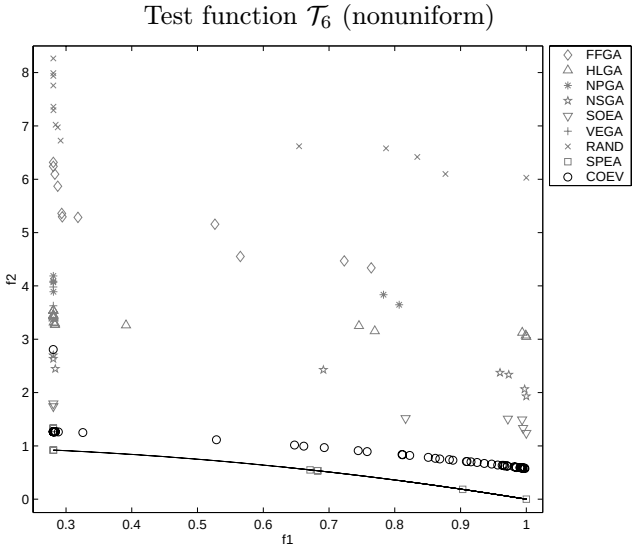


Fig. 4. Test functions $\mathcal{T}_4$ – $\mathcal{T}_6$.

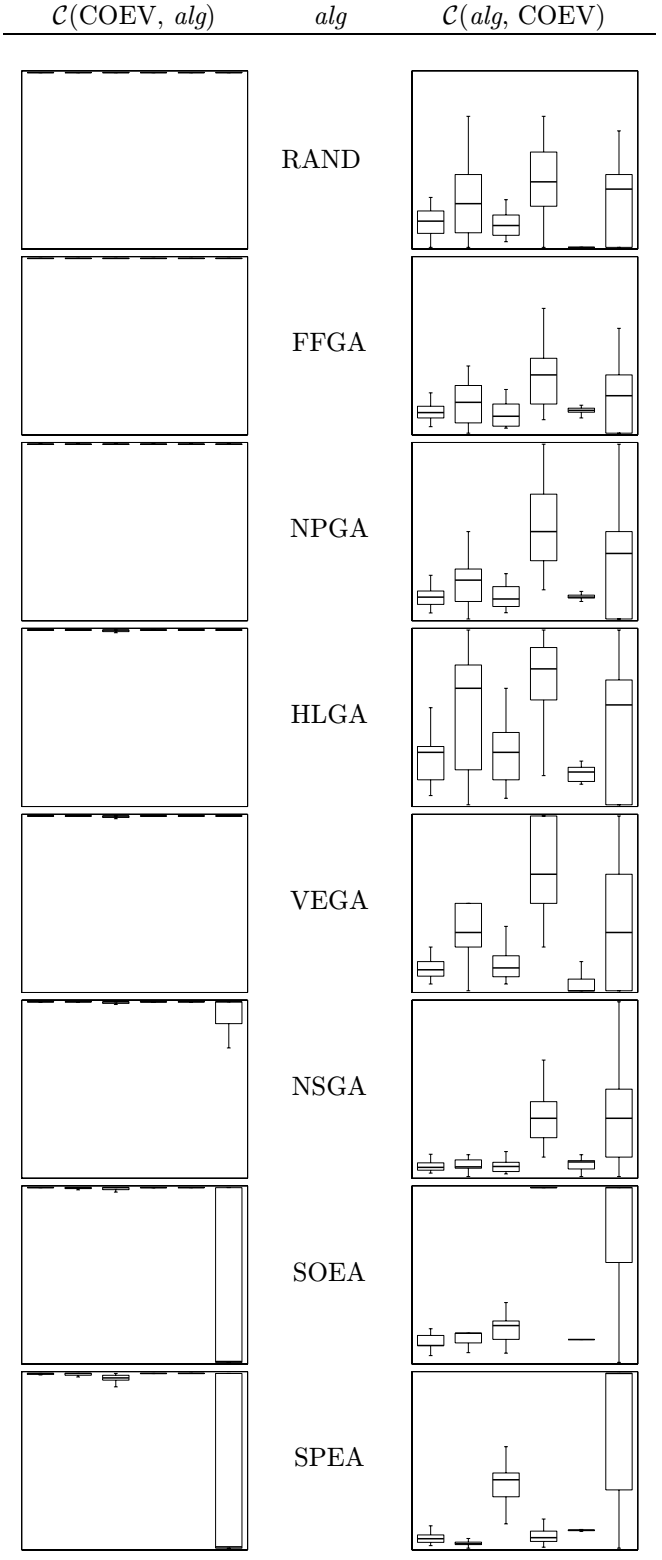


Fig. 5. Boxplots showing statistics from 30 samples of the function $\mathcal{C}$ comparing the CGA (COEV) to the other algorithms. Each boxplot contains results from each of the six test functions: the dark dash is the median, the the top of the box is the upper quartile, the bottom of the box is the lower quartile.